University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS 150 Fall 2002

Project Checkpoint #2
Displaying a Static Image Read from Memory on Screen

1 Objective

To read the contents of a Block RAM preloaded with data pertaining to a static image, and to display it on the monitors provided, using the ADV7194 Digital Video Encoder onboard.

2 Methodology

You are given a Block Ram which contains 2 lines of video data (i.e. 2*1440 bytes of data). Using this, you will display a picture which will resemble a series of color bars enclosed in a blue border. The first line of video from the BlockRam gives you video information required to draw the top and bottom lines of the border. The second line gives you the data to fill in the remaining lines.

Therefore, out of the 487 visible lines you have, The top 40 and bottom 40 lines should be filled with the border lines (line 0 from BlockRam), and the remaining lines filled up with data from line 1 of BlockRam.

To accomplish this, the sequence of tasks that are to be carried out are –

a. Initialize the ADV7194 by sending it I2C data to set a mode register
b. Start reading data from the BlockRam
c. Send the data multiplexed with control signals

The ports you will be concerned with for the purposes of this lab are :-

```
output [9:0] VE_P
output       VE_SCLK;
output       VE_SDA;
output       VE_PAL_NTSC;
output       VE_RESET_B;
output       VE_HSYNC_B;
output       VE_VSYNC_B;
output       VE_BLANK_B;
output       VE_SCRESET;
input        VE_CLOCK;
output       VE_CLKIN;
```

VE_CLOCK will be the 27MHz clock from the encoder which will be provided to you to work on.
VE_P are the 10 bits out of which you have to send your video and control data in the upper 8. It doesn't matter what you send in the lowest 2 bits. Ignore them.
VE_SCLK is the clock *you* will provide the encoder for *it* to synchronize with your I2C data being transmitted in
VE_SDA – I2C data output port

VE_PAL_NTSC – keep it to 0 for NTSC transmissions
VE_RESET_B – use it to RESET the encoder
VE_HSYNC_B, VE_VSYNC_B, VE_BLANK_B – not needed in our mode of operation. Tie it to logic 1
VE_SCRESET – not required. Ignore
VE_CLKIN – not required. Ignore

a. Initialization of the ADV7194

The first thing to be done is to RESET the ADV7194 with the PAL_NTSC pin pulled low. This sets all the registers, save one, to the state we want them to be to operate in our mode (mode 0-slave Ref. ADV7194Datasheet). Refer to the DataSheet for information on how pixel data is sent in mode 0 (Pg22). Using I^2C, then set Mode Register 2 to 60 Hex.

a1. Using I^2C

Read pages 28-29 of the ADV7194 Datasheet. The Address of the encoder is 01010100. The Address of the register we wish to write to is 00000010. During the acknowledgement phase, it doesn't matter which bit you send. Preferably send 0. It works at a maximum clock rate of 400KHz. Therefore use a counter to generate a clock of rate < 400KHz, and send the SCLK input that. Then start sending the data to SDA starting with the Start bit when the clock is high, and then the data bits. The order is :-
<Start><01010100><0><00000010><0><01100000><0><Stop>

Points to Note

- I^2C requires a clock speed less than 400KHz. Tone down the 27MHz clock you have.
- When sending data *make sure* SDA changes only when SCLK is low. If SDA changes when SCLK is high, it is seen as a control (Start/Stop) signal
- DO NOT CHANGE any registers other than mentioned here unless you are sure of what you are doing. Even though the datasheet mentions certain 'default' and 'recommended' values, the defaults are not entirely correct for our chip and what are recommended are not suited to our mode of operation. So listen to John Lennon and let it be.

In case you do wish to set more registers, you can either start/stop for each slave adder-regaddr-regdata sequence, or else just send slaveaddr-regaddr-regdata-(reg+1)data-(reg+2)data… continuously with the acknowledgement pauses in between.

b. Start reading data from the BlockRam

You are provided a file called **RamModule.v** which contains a module called **VideoMem**. This has your preloaded Block Ram. You are required to instantiate this module in yours and read the data. This module has the following ports: -

```
output[ 7:0] data;
```

```
input [10:0] ADDR;
input LINENO;
input CLK;
```

ADDR – can take values from 0-1439, corresponding to the 1440 bytes of data required to display one line of video.
LINENO – 0/1 – if 0, it returns you the line of video corresponding to the top/bottom border data. If 1, it returns you video data corresponding to the lines in between.
CLK – clock
Data – returns you one byte of pixel data.

The sequence of its sending you the pixel data is
ADDR[0] == Cb
ADDR[1] == Y
ADDR[2] == Cr
ADDR[3] == Y
Repeated until ADDR[1439] in that same sequence of Cb,Y,Cr,Y.

c. Send the data multiplexed with control signals

Once the I2C data is sent, start transmitting the Pixel data in the format shown in Pg22 of the Data sheet. Refer to VideoNutshell.doc for the sequence to be followed.

Suggested Design Methodology
The circuit basically has two states – one when it is sending the I2C data, and one when transmitting video data. You can also break it into further states of sending_eav, sending_horblank, sending_sav, sending_activideo. However, changing states usually loses one clock cycle (why?), and while transmitting video, we have to send a new byte every clock cycle. Therefore, it would be simpler to just have a counter running and let various operations like sending EAV, blanking data, SAV, video data take place depending on the value of the counter.
Whenever RESET is pressed, restart from the beginning, resending the I^2C data.

Points to Note

- It's not necessary to start transmitting your EAV the moment you finish transmitting the I^2C signal. Take your time. Transmit starting from anywhere in the bitstream. The video encoder will wait till it gets an EAV (ff, 00, 00, code) before it begins decoding your data. Just make sure you have your sequence down pat, as in immediately after the VBI gets sent transmitting the blue border, then the color lines, and then the blue border again. Also, once you begin sending, you have to maintain the sequence of EAV, HB, SAV, Video *exactly* with the exact number of bits.
- If you get a slightly shaky picture, then you are not sending 1716 (4+268+4+1440) bytes of data exactly. Check your state machine/counter logic.
- If you are getting weird colors, you probably are not reading the memory in 'phase' (i.e., Cb first, Y next, Cr, and then Y).

- Wrong values of Y,Cb and Cr could also result in the monitor losing sync! Therefore, if you want to play around with values, preferably start with the values given in the block ram files (RamInitSyn*.txt)
- The initial circuit for sending the I^2C will simulate, but once you add the RamModule, your simulation might not work for many cycles, as it takes too much memory.

3 Testing your I^2C.

To test whether your I^2C circuit is working properly, the Video Encoder has a register which can be set which will bring up a picture of color bars on screen (without borders, so you cannot pass that off as checkpoint2 ☺). The register to be set for this is register 4, and the value is 40Hex. So, since you are setting register 2 to 60, you can continue down the line, set register three to 00, and then 4 to 40. So, the sequence of bytes to be sent on I^2C would be
S<Encoder Addr>0< 02 Hex>0<60 Hex>0<00 Hex>0<40 Hex>S

You can simulate this by having a shift register which you load with the bitstream you want, and then shifting it out.

4 PreLab

- Design your circuit.
- Write the test bench for the I^2C simulation.

5 CheckPoint

The video as seen in the sample bit file provided should appear on the monitor S-Video display.

6 Acknowledgements

Original lab by Prof.John Wawryznek and N. Vinay Krishnan
Verilog and Block Ram expertise – Norm Zhou
Video Expertise – Tom Oberheim
Coffee for the late nights – Café Nefelli